Graph-based Nearest Neighbor Search: From Practice to Theory

Liudmila Prokhorenkova [*1 2 3] **Aleksandr Shekhovtsov** [*1 3]

Abstract

Graph-based approaches are empirically shown to be very successful for the nearest neighbor search (NNS). However, there has been very little research on their theoretical guarantees. We fill this gap and rigorously analyze the performance of graph-based NNS algorithms, specifically focusing on the low-dimensional ($d \ll \log n$) regime. In addition to the basic greedy algorithm on nearest neighbor graphs, we also analyze the most successful heuristics commonly used in practice: speeding up via adding shortcut edges and improving accuracy via maintaining a dynamic list of candidates. We believe that our theoretical insights supported by experimental analysis are an important step towards understanding the limits and benefits of graph-based NNS algorithms.

1. Introduction

Many methods in machine learning, pattern recognition, coding theory, and other research areas are based on nearest neighbor search (Bishop, 2006; Chen & Shah, 2018; May & Ozerov, 2015; Shakhnarovich et al., 2006). In particular, the k-nearest neighbor method is included in the list of top 10 algorithms in data mining (Wu et al., 2008). Since modern datasets are mostly massive (both in terms of the number of elements n and the dimension d), reducing the computation complexity of NNS algorithms is of the essence. The nearest neighbor problem is to preprocess a given dataset $\mathcal{D}$ in such a way that for an arbitrary forthcoming query vector q we can quickly (in time $o(n)$) find its nearest neighbors in $\mathcal{D}$.

Many efficient methods exist for NN problem, especially when the dimension d is small (Arya et al., 1998; Bentley, 1975; Meiser, 1993). In particular, the algorithms based on recursive partitions of the space, like k-d threes and random projection trees, are widely used (Bentley, 1975; Dasgupta & Freund, 2008; Dasgupta & Sinha, 2015; Keivani & Sinha, 2018). The most well-known algorithm usable for large d is the Locality Sensitive Hashing (LSH) (Indyk & Motwani, 1998), which is well studied theoretically and widely used in practical applications.

Recently, graph-based approaches were shown to demonstrate superior performance over other types of algorithms in many large-scale applications of NNS (Aumüller et al., 2019). Most graph-based methods are based on constructing a nearest neighbor graph (or its approximation), where nodes correspond to the elements of $\mathcal{D}$, and each node is connected to its nearest neighbors by directed edges (Dong et al., 2011; Hajebi et al., 2011; Wang et al., 2012). Then, for a given query q, one first takes an element in $\mathcal{D}$ (either random or fixed predefined) and makes greedy steps towards q on the graph: at each step, all neighbors of a current node are evaluated, and the one closest to q is chosen. Various additional heuristics were proposed to speed up graph-based search (Baranchuk et al., 2019; Fu et al., 2019; Iwasaki & Miyazaki, 2018; Malkov et al., 2014; Malkov & Yashunin, 2018).

While there is a lot of evidence empirically showing the superiority of graph-based NNS algorithms in practical applications, there is very little theoretical research supporting this. Laarhoven (2018) made the first step in this direction by providing time–space trade-offs for approximate nearest neighbor (ANN) search on sparse datasets uniformly distributed on a d-dimensional Euclidean sphere. The current work significantly extends these results and differs in several important aspects, as discussed in the next section.

Our analysis assumes the uniform distribution on a sphere, and we mostly focus on the dense regime $d \ll \log n$. This setup is motivated by our experiments demonstrating that *uniformization* and *densification* applied to a *general* dataset may improve some graph-based NNS algorithms. The dense regime is additionally motivated by the fact that real-world datasets are known to have low intrinsic dimension (Beygelzimer et al., 2006; Lin & Zhao, 2019).

[*]Equal contribution [1]Yandex, Moscow, Russia [2]Moscow Institute of Physics and Technology, Dolgoprudny, Moscow Region, Russia [3]Higher School of Economics, Moscow, Russia. Correspondence to: Liudmila Prokhorenkova <ostroumova-la@yandex-team.ru>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

We first investigate how the greedy search on simple NN graphs work in dense and sparse regimes (Section 4.2). For the dense regime, the time complexity is $\Theta\left(n^{1/d} \cdot M^d\right)$ for some constant M. Here M^d corresponds to the complexity of one step and $n^{1/d}$ to the number of steps.

Then, we analyze the effect of long-range links (Section 4.3): in practice, shortcut edges between distant elements are added to NN graphs to make faster progress in the early stages of the algorithm. This is a core idea of, e.g., navigable small-world graphs (NSW and HNSW) (Malkov et al., 2014; Malkov & Yashunin, 2018). While a naive method of adding long links uniformly at random does not improve the asymptotic query time, we show that properly added edges can reduce the number of steps to $O\left(\log^2 n\right)$. This part is motivated by Kleinberg (2000), who proved a related result for a greedy routing on a two-dimensional lattice. We adapt Kleinberg's result to NNS on a dataset uniformly distributed over a sphere in $\mathbb{R}^d$ (with possibly growing d). Additionally, while Kleinberg's theorem inspired many algorithms (Beaumont et al., 2007; Karbasi et al., 2015; Malkov & Yashunin, 2018), to the best of our knowledge, we are the first to propose an efficient method of constructing properly distributed long edges for a *general* dataset.

We also consider a heuristics known to significantly improve the accuracy: maintaining a dynamic list of several candidates instead of just one optimal point, which is a general technique known as *beam search*. We rigorously analyze the effect of this technique, see Section 4.4.

Finally, in Section 5, we empirically illustrate the obtained results, discuss the most promising techniques, and demonstrate why our assumptions are reasonable.

2. Related Work

Several well-known algorithms proposed for NNS are based on recursive partitions of the space, e.g., k-d trees and random projection trees (Bentley, 1975; Dasgupta & Freund, 2008; Dasgupta & Sinha, 2015; Keivani & Sinha, 2018). The query time of a k-d tree is $O\left(d \cdot 2^{O(d)}\right)$, which leads to an efficient algorithm for the exact NNS in the dense regime $d \ll \log n$ (Chen & Shah, 2018). When $d \gg \log n$, all algorithms suffer from the curse of dimensionality, so the problem is often relaxed to approximate nearest neighbor (ANN) formally defined in Section 3. Among the algorithms proposed for the ANN problem, LSH (Indyk & Motwani, 1998) is the most theoretically studied one. The main idea of LSH is to hash the points such that the probability of collision is much higher for points that are close to each other than for those that are far apart. Then, one can retrieve near neighbors for a query by hashing it and checking the elements in the same buckets. LSH solves c-ANN with query time $\Theta\left(dn^{\varphi}\right)$ and space complexity $\Theta\left(n^{1+\varphi}\right)$. Assuming the Euclidean distance, the optimal φ is about $\frac{1}{c^2}$ for data-agnostic algorithms (Andoni & Indyk, 2008; Datar et al., 2004; Motwani et al., 2007; O'Donnell et al., 2014). By using a data-dependent construction, this bound can be improved up to $\frac{1}{2c^2-1}$ (Andoni & Razenshteyn, 2015; Andoni & Razenstеyn, 2016). Becker et al. (2016) analyzed spherical locally sensitive filters (LSF) and showed that LSF can outperform LSH when the dimension d is logarithmic in the number of elements n. LSF can be thought of as lying in the middle between LSH and graph-based methods: LSF uses small spherical caps to define filters, while graph-based methods also allow, roughly speaking, moving to the neighboring caps.

In contrast to LSH, graph-based approaches are not so well understood. It is known that if we want to be able to find the exact nearest neighbor for any possible query via a greedy search, then the graph has to contain the classical Delaunay graph as a subgraph. However, for large d, Delaunay graphs have huge node degrees and cannot be constructed in a reasonable time (Navarro, 2002). In a more recent theoretical paper, Laarhoven (2018) considers datasets uniformly distributed on a d-dimensional sphere with $d \gg \log n$ and provides time–space trade-offs for the ANN search. The current paper, while using a similar setting, differs in several important aspects. First, we mostly focus on the dense regime $d \ll \log n$, show that it significantly differs both in terms of techniques and results, and argue why it is reasonable to work in this setting. Second, in addition to the analysis of plain NN graphs, we are the first to analyze how some additional tricks commonly used in practice affect the accuracy and complexity of the algorithm. These tricks are adding shortcut edges and using beam search. Finally, we support some claims made in the previous work by a rigorous analysis (see Supplementary Materials E). Let us also briefly discuss a recent paper (Fu et al., 2019), which rigorously analyzes the time and space complexity for searching the elements of so-called monotonic graphs. Note that the obtained guarantees are provided only for the case when a query coincides with an element of a dataset, which is a very strong assumption, and it is not clear how the results would generalize to a general setting.

A part of our research is motivated by Kleinberg (2000), who proved that after adding random edges with a particular distribution to a lattice, the number of steps needed to reach a query via a greedy routing scales polylogarithmically, see details in Section 4.3. We generalize this result to a more realistic setting and propose a practically applicable method to generate such edges.

Finally, let us mention a recent empirical paper comparing the performance of HNSW with other graph-based NNS algorithms (Lin & Zhao, 2019). In particular, it shows that HNSW has superior performance over one-layer graphs only

for low-dimensional data. We demonstrate theoretically why this can be the case (assuming uniform datasets): we prove that for plain NN graphs and for $d \gg \sqrt{\log n}$, the number of steps of graph-based NNS is negligible compared with one-step complexity. Moreover, for $d \gg \log n$, the algorithm converges in just two steps. Interestingly, Figure 5 in Lin & Zhao (2019) shows that on uniformly distributed synthetic datasets simple kNN graphs and HNSW show quite similar results, especially when the dimension is not too small. This means that studying simple NN graphs is a reasonable first step in the analysis of graph-based NNS algorithms. In contrast, on real datasets, HNSW is significantly better, which is caused by a so-called diversification of neighbors. However, as we show empirically in Section 5, simple kNN graphs can work comparably to HNSW even on real datasets, if we use the uniformization procedure (Sablayrolles et al., 2018).

3. Setup and Notation

We are given a dataset $\mathcal{D} = \{\mathbf{x}_1, \dots \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathbb{R}^{d+1}$ and assume that all elements of $\mathcal{D}$ belong to a unit Euclidean sphere, $\mathcal{D} \subset \mathcal{S}^d$. This special case is of particular importance for practical applications since feature vectors are often normalized.[1] For a given query $q \in \mathcal{S}^d$ let $\bar{\mathbf{x}} \in \mathcal{D}$ be its nearest neighbor. The aim of the exact NNS is to return $\bar{\mathbf{x}}$, while in c, R-ANN (approximate *near* neighbor), for given $R > 0$, $c > 1$, we need to find such $\mathbf{x}'$ that $\rho(q, \mathbf{x}') \le cR$ if $\rho(q, \bar{\mathbf{x}}) \le R$ (Andoni et al., 2017; Chen & Shah, 2018).[2] By $\rho(\cdot, \cdot)$ we further denote a spherical distance.

Similarly to Laarhoven (2018), we assume that the elements $\mathbf{x}_i \in \mathcal{D}$ are i.i.d. random vectors uniformly distributed on $\mathcal{S}^d$. Random uniform datasets are considered to be the most natural "hard" distribution for the ANN problem (Andoni & Razenshteyn, 2015). Hence, it is an important step towards understanding the limits and benefits of graph-based NNS algorithms.[3] From a practical point of view, real datasets are usually far from being uniformly distributed. However, in some applications uniformity is helpful, and there are techniques allowing to make a dataset more uniform while approximately preserving the distances (Sablayrolles et al., 2018). Remarkably, in our experiments (Section 5) we show that this trick, combined with dimensionality reduction, is beneficial for some graph-based NNS algorithms. We further assume that a query vector $q \in \mathcal{S}^d$ is placed uniformly within a distance R from the nearest neighbor $\bar{\mathbf{x}}$ (since c, R-ANN problem is defined conditionally on the event $\rho(q, \bar{\mathbf{x}}) \le R$). Such nearest neighbor is called *planted*.

In the current paper, we use a standard assumption that the dimensionality $d = d(n)$ grows with n (Chen & Shah, 2018). We distinguish three fundamentally different regimes in NN problems: dense with $d \ll \log(n)$; sparse with $d \gg \log(n)$; moderate with $d = \Theta(\log(n))$.[4] While Laarhoven (2018) focused solely on the sparse regime, we also consider the dense one, which is fundamentally different in terms of geometry, proof techniques, and query time.

As discussed in the introduction, most graph-based approaches are based on constructing a nearest neighbor graph (or its approximation). For uniformly distributed datasets, connecting an element $\mathbf{x}$ to a given number of nearest neighbors is essentially equivalent to connecting it to all such nodes $\mathbf{y}$ that $\rho(\mathbf{x}, \mathbf{y}) \le \rho^*$ with some appropriate ρ^* (since the number of nodes at a distance at most ρ^* is concentrated around its expectation, see also Supplementary Materials F.3 for an empirical illustration). Therefore, at the preprocessing stage, we choose some ρ^* and construct a graph using this threshold. Later, when we get a query q, we sample a random element $\mathbf{x} \in \mathcal{D}$ such that $\rho(\mathbf{x}, q) < \frac{\pi}{2}$,[5] and perform a graph-based greedy descent: at each step, we measure the distances between the neighbors of a current node and q and move to the closest neighbor, while we make progress. In the current paper, we assume one iteration of this procedure, i.e., we do not restart the process several times.

4. Theoretical Analysis

In this section, we overview the obtained theoretical results. We start with a preliminary analysis of the properties of spherical caps and their intersections. These properties give some intuition and are extensively used throughout the proofs. Then, we analyze the performance of the greedy search over plain NN graphs. We mostly focus on the dense regime, but also formulate the corresponding theorem for the sparse one. After that, we analyze how the shortcut edges affect the complexity of the algorithm. We prove that they indeed improve the asymptotic query time, but only in the dense regime. Finally, we analyze the positive effect of the beam search, which is widely used in practice.

[1] From a theoretical point of view, there is a reduction of ANN in the entire Euclidean space to ANN on a sphere (Andoni & Razenshteyn, 2015; Valiant, 2015).

[2] There is also a notion of c-ANN, where the goal is to find such $\mathbf{x}'$ that $\rho(q, \mathbf{x}') \le c\rho(q, \bar{\mathbf{x}})$; c-ANN can be reduced to c, R-ANN with additional $O(\log n)$ factor in query time and $O(\log^2 n)$ in storage cost (Chen & Shah, 2018; Har-Peled et al., 2012).

[3] Also, Andoni & Razenshteyn (2015) show how to reduce ANN on a generic dataset to ANN on a "pseudo-random" dataset for a data-dependent LSH algorithm.

[4] Throughout the paper $\log$ is a binary logarithm (with base 2) and $\ln$ is a natural logarithm (with base e).

[5] We can easily achieve this by trying a constant number of random samples since each sample succeeds with probability $1/2$. This trick speeds up the procedure (without affecting the asymptotics) and simplifies the proofs.

4.1. Auxiliary Results

Let us discuss some technical results on the volumes of spherical caps and their intersections, which we extensively use in the proofs.

Denote by μ the Lebesgue measure over $\mathbb{R}^{d+1}$. By $C_{\mathbf{x}}(\gamma)$ we denote a spherical cap of *height* γ centered at $\mathbf{x} \in \mathcal{S}^d$, i.e., $\{\mathbf{y} \in \mathcal{S}^d : \langle \mathbf{x}, \mathbf{y} \rangle \geq \gamma\}$; $C(\gamma) = \mu\left(C_{\mathbf{x}}(\gamma)\right)$ denotes the volume of a spherical cap of height γ. Throughout the paper, for any variable γ, $0 \leq \gamma \leq 1$, we let $\hat{\gamma} := \sqrt{1-\gamma^2}$. We say that $\hat{\gamma}$ is the *radius* of a spherical cap.

The volume of a spherical cap defines the expected number of neighbors of a node. We estimate this volume in Supplementary Materials A.1. Despite some additional terms (which can often be neglected), one can think that $C(\gamma) \propto \hat{\gamma}^d$.

The intersections of spherical caps are also important: their volumes are needed to estimate the probably of making a step of graph-based search. Formally, by $W_{\mathbf{x},\mathbf{y}}(\alpha, \beta)$ we denote the intersection of two spherical caps centered at $\mathbf{x} \in \mathcal{S}^d$ and $\mathbf{y} \in \mathcal{S}^d$ with heights α and β, respectively, i.e., $W_{\mathbf{x},\mathbf{y}}(\alpha, \beta) = \{\mathbf{z} \in \mathcal{S}^d : \langle \mathbf{z}, \mathbf{x} \rangle \geq \alpha, \langle \mathbf{z}, \mathbf{y} \rangle \geq \beta\}$. By $W(\alpha, \beta, \theta)$ we denote the volume of such intersection given that the angle between $\mathbf{x}$ and $\mathbf{y}$ is θ. In Supplementary Materials A.2, we estimate $W(\alpha, \beta, \theta)$. We essentially prove that for $\gamma = \frac{\sqrt{\alpha^2+\beta^2-2\alpha\beta\cos\theta}}{\sin\theta}$, $W(\alpha, \beta, \theta)$ (or its complement) $\propto \hat{\gamma}^d$.

Although our results are similar to those formulated by Becker et al. (2016), it is crucial for our analysis that parameters defining spherical caps depend on d and either γ or $\hat{\gamma}$ may tend to zero. In contrast, the results of Becker et al. (2016) hold only for fixed parameters. Also, we extend Lemma 2.2 in their paper by analyzing both $W(\alpha, \beta, \theta)$ and its complement: we need a lower bound on $W(\alpha, \beta, \theta)$ to show that with high probability we can make a step of the algorithm and an upper bound on $C(\alpha) - W(\alpha, \beta, \theta)$ to show that at the final step of the algorithm we can find the nearest neighbor with high probability.

The fact that $C(\gamma) \propto \hat{\gamma}^d$ allows us to understand the principal difference between the dense and sparse regimes. For dense datasets, we assume $d = d(n) = \log(n)/\omega$, where $\omega = \omega(n) \to \infty$ as $n \to \infty$. In this case, it is convenient to operate with radii of spherical caps. An essential property of the dense regime is the fact that the distance from a given point to its nearest neighbor behaves as $2^{-\omega}$, so it decreases with n. Indeed, let $\hat{\alpha}_1$ be the radius of a cap centered at a given point and covering its nearest neighbor, then we have $C(\alpha_1) \sim \frac{1}{n}$, i.e., $\hat{\alpha}_1 \sim n^{-\frac{1}{d}} = 2^{-\omega}$. To construct a nearest neighbor graph, we use spherical caps of radius $M \cdot 2^{-\omega}$ with some constant $M > 1$.

In sparse regime, we assume $d = \omega \log(n)$, $\omega \to \infty$ as $n \to \infty$. In this case, it is convenient to operate with heights of spherical caps. A crucial feature of sparse regime is that the heights under consideration tend to zero as n grows. Informally speaking, all nodes are at spherical distance about $\pi/2$ from each other. Indeed, we have $C(\alpha_1) \sim \frac{1}{n}$, i.e., $\alpha_1^2 \sim 1 - n^{-\frac{2}{d}} = 1 - 2^{-\frac{2}{\omega}} \propto w^{-1}$ and it tends to zero with n. In this case, to construct a graph, we use spherical caps with height equal to $\sqrt{M} \cdot \alpha_1$, where $M < 1$ is some constant.

4.2. Greedy Search on Plain NN Graphs

Dense regime For any constant $M > 1$, let $G(M)$ be a graph obtained by connecting $\mathbf{x}_i$ and $\mathbf{x}_j$ iff $\rho(\mathbf{x}_i, \mathbf{x}_j) \leq \arcsin\left(M\, n^{-1/d}\right)$. The following theorem is proven in Supplementary Materials B.1-B.3.

Theorem 1. *Assume that $d \gg \log\log n$ and we are given some constant $c \geq 1$. Let M be a constant such that $M > \sqrt{\frac{4c^2}{3c^2-1}}$, then, with probability $1-o(1)$, $G(M)$-based NNS solves c, R-ANN for any R (or the exact NN problem if $c = 1$); time complexity is $\Theta\left(d^{1/2} \cdot n^{1/d} \cdot M^d\right) = n^{o(1)}$; space complexity is $\Theta\left(n \cdot d^{-1/2} \cdot M^d \cdot \log n\right) = n^{1+o(1)}$.*

It follows from Theorem 1 that both time and space complexities increase with M (for constant M), so one may want to choose the smallest possible M. When the aim is to find the exact nearest neighbor ($c = 1$), we can take any $M > \sqrt{2}$. When $c > 1$, the lower bound for M decreases with c.

The space complexity straightforwardly depends on M: the radius of a spherical cap defines the expected number of neighbors for a node, which is $\Theta(d^{-1/2} \cdot M^d)$, and additional $\log n$ corresponds to storing integers up to n. The time complexity consists of two terms: the complexity of one step is multiplied by the number of steps. The complexity of one step is the number of neighbors multiplied by d: $\Theta(d^{1/2} \cdot M^d)$. The number of steps is $\Theta\left(n^{1/d}\right)$.

While d is negligible compared with M^d, the relation between $n^{1/d}$ and M^d is non-trivial. Indeed, when $d \gg \sqrt{\log n}$, the term M^d dominates, so in this regime, the smaller M the better asymptotics we get (both for time and space complexities). However, when the dataset is very dense, i.e., $d \ll \sqrt{\log n}$, the number of steps becomes much larger than the complexity of one step. For such datasets, it could be possible that taking $M = M(n) \gg 1$ would improve the query time (as it affects the number of steps). However, in Supplementary Materials B.4, we prove that this is not the case, and the query time from Theorem 1 cannot be improved.

Finally, since all distances considered in the proof tend to zero with n, it is easy to verify that all results stated above for the spherical distance also hold for the Euclidean one.

Sparse regime For any M, $0 < M < 1$, let $G(M)$ be a graph obtained by connecting $\mathbf{x}_i$ and $\mathbf{x}_j$ iff $\rho(\mathbf{x}_i, \mathbf{x}_j) \le \arccos\left(\sqrt{\frac{2M \ln n}{d}}\right)$. The following theorem holds (see Supplementary Materials B.5 for the proof).

Theorem 2. *For any $c > 1$ let $\alpha_c = \cos\left(\frac{\pi}{2c}\right)$ and let M be any constant such that $M < \frac{\alpha_c^2}{\alpha_c^2+1}$. Then, with probability $1-o(1)$, $G(M)$-based NNS solves c, R-ANN (for any R and for spherical distance); time complexity of the procedure is $\Theta\left(n^{1-M+o(1)}\right)$; space complexity is $\Theta\left(n^{2-M+o(1)}\right)$.*

Interestingly, as follows from the proof, in the sparse regime, the greedy algorithm converges in at most two steps with probability $1 - o(1)$ (on a uniform dataset). As a result, there is no trade-off between time and space complexity: larger values of M reduce both of them.

One can easily obtain an analog of Theorem 2 for the Euclidean distance. In Theorem 2, α_c is the height of a spherical cap covering a spherical distance $\frac{\pi}{2c}$, which is c times smaller than $\pi/2$. For the Euclidean distance, we have to replace $\frac{\pi}{2}$ by $\sqrt{2}$ and then the height of a spherical cap covering Euclidean distance $\sqrt{2}/c$ is $\alpha_c = 1 - \frac{1}{c^2}$. So, we get the following corollary.

Corollary 1. *For the Euclidean distance, Theorem 2 holds with $\alpha_c = 1 - \frac{1}{c^2}$, i.e., $M < \frac{\left(1-1/c^2\right)^2}{(1-1/c^2)^2+1}$.*

As a result, we can obtain time complexity n^{φ} and space complexity $n^{1+\varphi}$, where φ can be made about $\frac{1}{(1-1/c^2)^2+1}$. Note that this result corresponds to the case $\rho_q = \rho_s$ in Laarhoven (2018).

4.3. Analysis of Long-range Links

As discussed in the previous section, when $d \gg \sqrt{\log n}$, the number of steps is negligible compared to the one-step complexity. Hence, reducing the number of steps cannot change the main term of the asymptotics.[6] However, if $d \ll \sqrt{\log n}$ (very dense setting), the number of steps becomes the main term of time complexity. In this case, it is reasonable to reduce the number of steps via adding so-called long-range links (or shortcuts) — some edges connecting elements that are far away from each other — which may speed up the search on early stages of the algorithm.

The simplest way to obtain a graph with a small diameter from a given graph is to connect each node to a few neighbors chosen uniformly at random. This idea is proposed by Watts & Strogatz (1998) and gives $O(\log n)$ diameter for the so-called "small-world model". However, having a logarithmic diameter does *not* guarantee a logarithmic number of steps in graph-based NNS, since these steps, while being greedy in the underlying metric space, may not be optimal on a graph (Kleinberg, 2000). In Supplementary Materials C.1, we formally prove that adding edges uniformly at random cannot improve the asymptotic time complexity. The intuition is simple: choosing the closest neighbor among the long-range ones is equivalent to sampling a certain number of nodes (uniformly at random among the whole set) and then choosing the one closest to q.

This agrees with Kleinberg (2000), who considered a 2-dimensional grid supplied with some random long-range edges. Kleinberg assumed that in addition to the local edges, each node creates one random outgoing long link, and the probability of a link from u to v is proportional to $\rho(u, v)^{-r}$. He proved that for $r = 2$, the greedy graph-based search finds the target element in $O\left(\log^2 n\right)$ steps, while any other r gives at least n^{φ} with $\varphi > 0$. This result can be easily extended to *constant* $d > 2$, in this case one should take $r = d$ to achieve polylogarithmic number of steps.

Kleinberg (2000) influenced a large number of further studies. Some works generalized the result to other settings (Barrière et al., 2001; Bonnet et al., 2007; Duchon et al., 2006), others used it as a motivation of search algorithms (Beaumont et al., 2007; Karbasi et al., 2015). It is also mentioned as a motivation for a widely used HNSW algorithm (Malkov & Yashunin, 2018). However, Kleinberg's probabilities have not been used directly since 1) it is unclear how to use them for general distributions, when the intrinsic dimension is not known or varies over the dataset, 2) generating properly distributed edges has $\Theta\left(n^2 d\right)$ complexity, which is infeasible for many practical tasks.

We address these issues. First, we translate the result of Kleinberg (2000) to our setting (importantly, we have $d \to \infty$). Second, we show how one can apply the method to general datasets without thinking about the intrinsic dimension and non-uniform distributions. Finally, we discuss how to reduce the computational complexity of graph construction procedure.

Following Kleinberg (2000), we draw long-range edges with the following probabilities:

$$\mathrm{P}(\text{edge from } u \text{ to } v) = \frac{\rho(u, v)^{-d}}{\sum_{w \neq u} \rho(u, w)^{-d}}. \qquad (1)$$

Theorem 3. *Under the conditions of Theorem 1, sampling one long-range edge for each node according to (1) reduces the number of steps to $O(\log^2 n)$ (with probability $1-o(1)$).*

This theorem is proven in Supplementary Materials C.2. It is important that in contrast to Kleinberg (2000), we assume $d \to \infty$. Indeed, a straightforward generalization of Kleinberg's result to non-constant d gives an additional 2^d

[6] This agrees with the empirical results obtained by Lin & Zhao (2019), where on synthetic datasets the difference between simple kNN graphs and the more advanced HNSW algorithm becomes smaller as d increases and vanishes after $d = 8$.

multiplier, which we were able to remove.

Note that using long-range edges, we can guarantee $O\left(\log^2 n\right)$ steps, while plain NN graphs give $\Theta\left(n^{1/d}\right)$ (see Theorem 1 and the discussion after that). Hence, reducing the number of steps is reasonable if $\log^2 n < n^{1/d}$, which means that $d < \frac{\log n}{2 \log \log n}$.

As follows from the proof, adding more long-range edges may further reduce the number of steps. It is theoretically shown in the following corollary and is empirically evaluated in Supplementary Materials F.3.

Corollary 2. *If for each node we add $\Theta(\log n)$ long-range edges, then the number of steps becomes $O(\log n)$, so the time complexity is $O\left(d^{1/2} \cdot \log n \cdot M^d\right)$ while the asymptotic space complexity does not change compared to Theorem 1. Further increasing the number of long-range edges does not improve the asymptotic complexity.*

Also, we suggest the following trick, which noticeably reduces the number of steps in practice while not affecting the theoretical asymptotics: at each iteration, we check the long-range links first and proceed with the local ones only if we cannot make progress with long edges. We empirically evaluate this trick (called LLF) in Section 5.

It is non-trivial how to apply Theorem 3 in practice due to the dependence of probabilities on d: real datasets usually have a low intrinsic dimension even when embedded to a higher-dimensional space (Beygelzimer et al., 2006; Lin & Zhao, 2019), and the intrinsic dimension may vary over the dataset. Let us show how to make the distribution in (1) dimension-free. For lattices and uniformly distributed datasets, the distance to a node is strongly related to the *rank* of this node in the list of all elements, when they ordered by a distance. Formally, let v be the k-th neighbor of u. Then, we define $\rho_{rank}(u, v) = (k/n)^{1/d}$. For uniform datasets, $\rho(u, v)$ locally behaves as $\rho_{rank}(u, v)$ (up to a constant multiplier) since the number of nodes at a distance ρ from a given one grows as ρ^d. If we replace ρ by ρ_{rank} in (1), we obtain:

$$\mathrm{P}(\text{edge to } k\text{-th neighbor}) = \frac{1/k}{\sum_{i=1}^{n} 1/i} \sim \frac{1}{k \ln n}. \quad (2)$$

This distribution is *dimension independent*, i.e., it can be easily used for general datasets.

Finally, we address the problem of the computational complexity of graph construction procedure. For this, we propose to generate long edges as follows: for each source node, we first choose n^φ, $0 < \varphi < 1$, candidates uniformly at random, and then sample a long edge from this set according to the probabilities in (2). Pre-sampling reduces the time complexity to $\Theta\left(n^{1+\varphi}\ (d + \log n^\varphi)\right)$, compared with $\Theta(n^2 d)$ in Theorem 3. The following lemma shows how it affects the distribution.

Lemma 1. *Let P_k be the probability defined in (2) and P_k^φ be the corresponding probability assuming the pre-sampling of n^φ elements. Then, for all $k > n^{1-\varphi}$, $P_k^\varphi / P_k = \Theta(1/\varphi)$.*

Informally, Lemma 1 says that for most of the elements, the probability does not change significantly. In Section 5, we use pre-sampling with $\varphi = 1/2$ and in Supplementary Materials F.3 we show that this pre-sampling and using (2) instead of (1) does not affect the quality of graph-based NNS.

4.4. Beam Search

Beam search is a heuristic algorithm that explores a graph by expanding the most promising element in a limited set. It is widely used in graph-based NNS algorithms (Fu et al., 2019; Malkov & Yashunin, 2018) as it drastically improves the accuracy. Moreover, it was shown that even a simple kNN graph supplied with beam search can show results close to the state-of-the-art on synthetic datasets (Lin & Zhao, 2019). However, to the best of our knowledge, we are the first to analyze the effect of beam search in graph-based NNS algorithms theoretically.

Theorem 1 states that to solve the exact NN problem with probability $1 - o(1)$, we need to have about M^d neighbors for each node, where $M > \sqrt{2}$. If we reduce M below this bound, then with a significant probability, the algorithm may get stuck in a local optimum before it reaches the nearest neighbor. As follows from the proof, the problem of local optima is critical for the last steps of the algorithm, i.e., large degrees are needed near the query.

Let us show that M can be reduced if beam search is used instead of greedy search. Assume that we have reached some local neighborhood of the query q and there are $m - 1$ points closer to q in the dataset. Further, assume that we use beam search with at least m best candidates stored. If the subgraph on m nodes closest to q is connected, then after at most m steps the beam search terminates and returns m closest nodes including the nearest neighbor (see Figure 1 for an illustration). Thus, we can reduce the size of the neighborhood, but instead of getting stuck in a local optimum, we explore the neighborhood until we reach the target. Formally, the following theorem holds.

Theorem 4. *Let $M > 1$, $L > 1$ be such constants that $M^2\left(1 - \frac{M^2}{4L^2}\right) > 1$ and let $\log \log n \ll d \ll \log n$. Assume that we use beam search with $\frac{C\,L^d}{\sqrt{d}}$ candidates (for a sufficiently large C) and we add $\Theta(\log n)$ long-range edges. Then, $G(M)$-based NNS solves the exact NN problem with probability $1 - o(1)$. The time complexity is $O\left(L^d \cdot M^d\right)$.*

As a result, beam search allows to significantly reduce degrees of a graph, which finally leads to time complexity

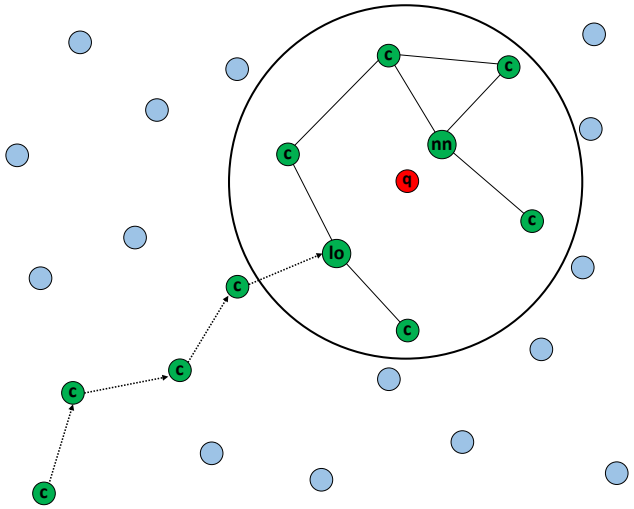

Figure 1. Example of beam search: "q" — query, "nn" — nearest neighbor, "lo" — local optimum for greedy search, "c" — other candidates visited during NNS. Beam search with 7 candidates returns 7 nearest elements to the query since the subgraph induced by them is connected.

reduction. We empirically illustrate this fact in Table 2 in the next section.

Theorem 4 gives the time–space trade-off for the beam search: we can reduce M, which determines the space complexity, to any value grater than 1, but for small M we have to take large L which increases the query time. The following corollary optimizes the time complexity.

Corollary 3. *In Theorem 4, we can take* $M = \sqrt{\frac{3}{2}}$ *and any* $L > \sqrt{\frac{9}{8}}$. *As a result, the main term of the time complexity can be reduced to* $\left(\frac{27}{16}\right)^{d/2}$, *which is less than* $2^{d/2}$ *from Theorem 1.*

5. Experiments

In this section, we illustrate the obtained results using synthetic datasets and also analyze whether our findings translate to real data. We also demonstrate how uniformization and densification may improve the quality of some graph-based NNS algorithms, which motivates the assumptions used in our analysis.

5.1. Experimental Setup

Datasets To illustrate our theoretical results, we generated synthetic datasets uniformly distributed over d-dimensional spheres with $n = 10^6$ and $d \in \{2, 4, 8, 16\}$. We also experimented with the widely used real-world datasets: SIFT and GIST (Jegou et al., 2010), GloVe (Pennington et al., 2014), and DEEP1B (Babenko & Lempitsky, 2016). These datasets are summarized in Table 1.

Table 1. Real datasets

dataset	dim	# base	# queries	metric
SIFT	128	10^6	10^4	Euclidean
GIST	960	10^6	10^3	Euclidean
GloVe	300	10^6	10^4	Angular
DEEP	96	10^6	10^4	Euclidean

Table 2. KNN graphs on synthetic data

dim	steps	degree	Recall@1	Beam
2	200	20	0.998	1
4	15	60	0.999	1
8	5	300	0.998	1
16	4	2000	0.99	1
16	106	20	0.995	100

Measuring performance To evaluate the algorithms, we adopt a standard approach and compare their Time-versus-Quality curves. On x-axis, instead of a frequently used Recall@1, we have $\text{error} = 1 - \text{Recall@1}$ and use the logarithmic scale for better visualisation, since the most important region is where error is close to zero. On y-axis, for synthetic datasets, we show the number of distance calculations (the smaller the better), since we want to illustrate our theoretical results. For real datasets, we measure the number of queries per second (the larger the better). All algorithms were run on a single core of Intel(R) Core(TM) i7-6800K CPU @ 3.40GHz.

Algorithms In the experiments, we combine and analyze the techniques discussed in this paper:

- KNN is a simple NN graph, where each node has a fixed number of neighbors;
- KL means that for each node we additionally add edges distributed according to (2), we also use pre-sampling of $\sqrt{n}$ nodes to speed up the construction procedure;
- LLF: at each iteration check *long links first* and consider local neighbors only if required;
- BEAM: use beam search instead of greedy search, always used on real data;
- DIM-RED: dimensionality reduction technique used on real datasets, as discussed in more details in Section 5.3.

To obtain Time-vs-Quality curves, we vary the number of

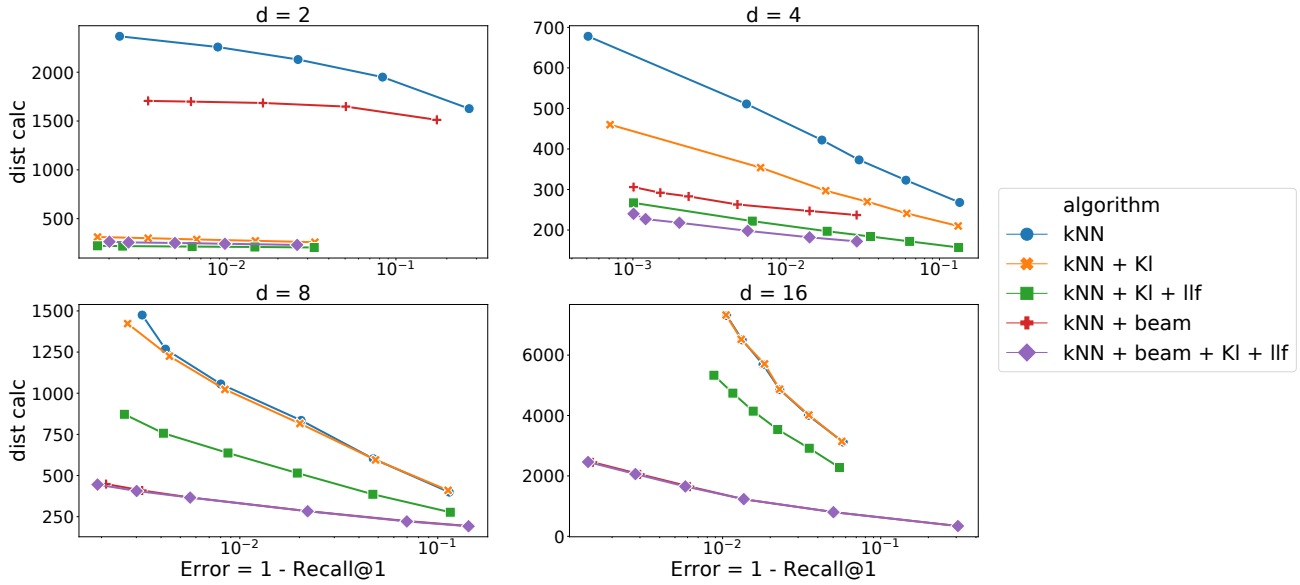

Figure 2. Results on synthetic datasets

local neighbors for greedy search and the number of candidates for beam search.

For reference, on real datasets, we compare all algorithms with HNSW (Malkov & Yashunin, 2018), as it is shown to provide the state-of-the-art performance on the common benchmarks and its source code is publicly available.

5.2. Synthetic Datasets

In this section, we illustrate our theoretical results from Section 4 on synthetic datasets. The source code of these experiments is publicly available.[7]

Figure 2 shows plain NN graphs (Theorems 1 and 2), the effect of adding long-range links (Theorem 3), the LLF heuristics, and the beam search (Theorem 4).

In small dimensions, the largest improvement is caused by using properly distributed long edges. However, for larger d, adding such edges does not help much, which agrees with our theoretical results: when d is large, the number of steps becomes negligible compared to the complexity of one step (see Section 4.3). Note that LLF always improves the performance, which is expected, since LLF reduces the number of distance computations for local neighbors.

In contrast, the effect of BEAM search is relatively small for $d = 2$, and it becomes substantial as d grows. This agrees with our analysis in Section 4.4: beam search helps to reduce the degrees, which are especially large for sparse datasets. Indeed, Theorems 1 and 2 show that graph degrees are expected to explode with d. Table 2 additionally illustrates this: we fix a sufficiently high value of Recall@1 and analyze which values of k would allow KNN to reach such performance (approximately). We see that degrees explode: while 20 is sufficient for $d = 2$, we need 2000 neighbors for $d = 16$. In contrast, the number of steps decreases with d from 200 to 4, which also agrees with our analysis. However, using the beam search with size 100 reduces the number of neighbors back to 20 while still having fewer steps compared to $d = 2$.

5.3. Real Datasets

While our theoretical results assume uniformly distributed elements, in this section we analyze their viability in real settings.

Let us first discuss a technique (DIM-RED) that exploits uniformization and densification and allows us to improve the quality of KNN significantly. This technique is inspired by our theoretical results and also by the observation that on uniform datasets for $d > 8$ plain NN graphs and HNSW have similar performance (Lin & Zhao, 2019). The basic idea is to map a given dataset to a smaller dimension and at the same time make it more uniform while trying to preserve the neighborhoods. For this purpose, we use the technique proposed by Sablayrolles et al. (2018). At the preprocessing step, we construct a search graph on the new (smaller-dimensional) dataset. This dataset is dense and close to uniform, so our theoretical results are applicable. Then, for a given query q, we obtain a lower-dimensional q' via the same transformation and then perform the beam

[7] https://github.com/Shekhale/gbnns_theory

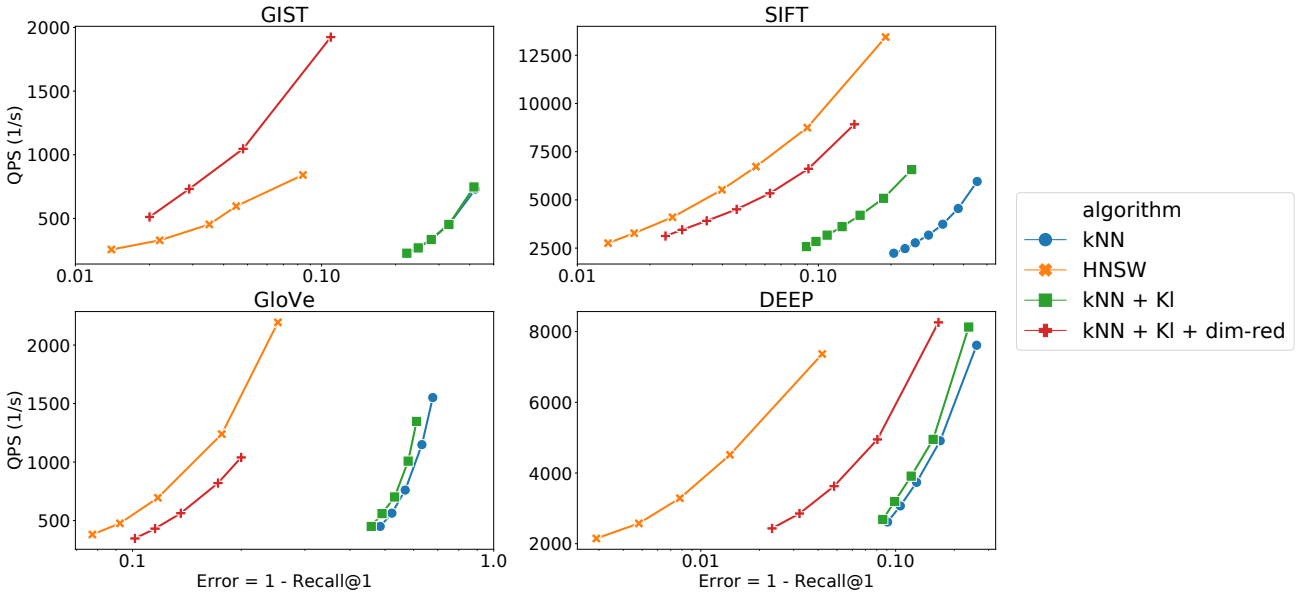

Figure 3. Results on real datasets

search there. Finally, we come back to the original space and find the element closets to the original query from the beam candidates. This technique is called DIM-RED.

We compare all algorithms in Figure 3. Recall that in all cases we use BEAM search since it drastically improves the greedy algorithm. We observe that long edges significantly improve the performance of KNN (in the original space) only for the SIFT dataset. This agrees with our analysis: SIFT has the smallest intrinsic dimension (Lin & Zhao, 2019). However, combining the algorithm with DIM-RED, we get a significant speedup. For GIST, the obtained algorithm is even better than HNSW. This improvement may be caused by the fact that the original dimension is excessive, so we may reduce it without affecting the neighborhoods much. A smaller dimensionality is beneficial since it allows for faster distance computations and may also help to avoid local optima which are critical for sparse datasets.

These results confirm that analysis of uniform distributions and dense datasets is important: while these conditions are not satisfied in real datasets, they are close to being satisfied after the transformation.

The source code for reproducing these experiments is publicly available.[8] Also, in concurrent research, we show that with some additional modifications DIM-RED can be used to obtain a new state-of-the-art graph-based NNS algorithm.

[8] https://github.com/Shekhale/gbnns_dim_red

6. Conclusion

In this paper, we theoretically analyze the performance of graph-based NNS, specifically focusing on the dense regime $d \ll \log(n)$. We make an important step from practice to theory: in addition to plain NN graphs, we also analyze the effect of two heuristics widely used in practice: shortcut edges and beam search.

Since graph-based NNS algorithms become extremely popular nowadays, we believe that more theoretical analysis of such methods will follow. A natural direction for future research would be to find broader conditions under which similar guarantees can be obtained. In particular, in the current research, we assume uniform distribution. We supported this assumption empirically, but clearly, the analysis of more general distributions would be useful. While our results can be straightforwardly extended to distributions similar to uniform (e.g., if the density varies in some bounded interval), further generalizations seem to be tricky. Another promising direction is to analyze the effect of diversification (Harwood & Drummond, 2016), which was empirically shown to improve the quality of graph-based NNS (Lin & Zhao, 2019). However, this is reasonable to do in a more general setting, since diversification is especially important for non-uniform distributions.

Acknowledgments

The authors thank Artem Babenko, Dmitry Baranchuk, and Stanislav Morozov for fruitful discussions.

References

Andoni, A. and Indyk, P. Near-optimal hashing algorithms for near neighbor problem in high dimension. *Communications of the ACM*, 51(1):117–122, 2008.

Andoni, A. and Razenshteyn, I. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 793–801. ACM, 2015.

Andoni, A. and Razensteyn, I. Tight lower bounds for data-dependent locality-sensitive hashing. In *32nd International Symposium on Computational Geometry (SoCG 2016)*, 2016.

Andoni, A., Laarhoven, T., Razenshteyn, I., and Waingarten, E. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 47–66. Society for Industrial and Applied Mathematics, 2017.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

Aumüller, M., Bernhardsson, E., and Faithfull, A. ANN-benchmarks: a benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 2019.

Babenko, A. and Lempitsky, V. Efficient indexing of billion-scale datasets of deep descriptors. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Baranchuk, D., Persiyanov, D., Sinitsin, A., and Babenko, A. Learning to route in similarity graphs. In *International Conference on Machine Learning*, pp. 475–484, 2019.

Barrière, L., Fraigniaud, P., Kranakis, E., and Krizanc, D. Efficient routing in networks with long range contacts. In *International Symposium on Distributed Computing*, pp. 270–284. Springer, 2001.

Beaumont, O., Kermarrec, A.-M., and Rivière, É. Peer to peer multidimensional overlays: approximating complex structures. In *International Conference On Principles Of Distributed Systems*, pp. 315–328. Springer, 2007.

Becker, A., Ducas, L., Gama, N., and Laarhoven, T. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pp. 10–24, 2016.

Bentley, J. L. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18 (9):509–517, 1975.

Beygelzimer, A., Kakade, S., and Langford, J. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pp. 97–104, 2006.

Bishop, C. M. *Pattern recognition and machine learning*. Springer, 2006.

Bonnet, F., Kermarrec, A.-M., and Raynal, M. Small-world networks: is there a mismatch between theory and practice? 2007.

Chen, G. H. and Shah, D. Explaining the success of nearest neighbor methods in prediction. *Foundations and Trends® in Machine Learning*, 10(5-6):337–588, 2018.

Dasgupta, S. and Freund, Y. Random projection trees and low dimensional manifolds. In *STOC*, volume 8, pp. 537–546. Citeseer, 2008.

Dasgupta, S. and Sinha, K. Randomized partition trees for nearest neighbor search. *Algorithmica*, 72(1):237–263, 2015.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262. ACM, 2004.

Dong, W., Moses, C., and Li, K. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pp. 577–586. ACM, 2011.

Duchon, P., Hanusse, N., Lebhar, E., and Schabanel, N. Could any graph be turned into a small-world? *Theoretical Computer Science*, 355(1):96–103, 2006.

Fu, C., Xiang, C., Wang, C., and Cai, D. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment*, 12(5): 461–474, 2019.

Hajebi, K., Abbasi-Yadkori, Y., Shahbazi, H., and Zhang, H. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

Har-Peled, S., Indyk, P., and Motwani, R. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

Harwood, B. and Drummond, T. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5713–5722, 2016.

Indyk, P. and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613. ACM, 1998.

Iwasaki, M. and Miyazaki, D. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *arXiv preprint arXiv:1810.07355*, 2018.

Jegou, H., Douze, M., and Schmid, C. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

Karbasi, A., Ioannidis, S., and Massoulié, L. From small-world networks to comparison-based search. *IEEE Transactions on Information Theory*, 61(6):3056–3074, 2015.

Keivani, O. and Sinha, K. Improved nearest neighbor search using auxiliary information and priority functions. In *International Conference on Machine Learning*, pp. 2578–2586, 2018.

Kleinberg, J. The small-world phenomenon: an algorithmic perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pp. 163–170, 2000.

Laarhoven, T. Graph-based time-space trade-offs for approximate near neighbors. In *34th International Symposium on Computational Geometry (SoCG 2018)*, 2018.

Lin, P.-C. and Zhao, W.-L. Graph based nearest neighbor search: promises and failures. *arXiv. org*, 2019.

Malkov, Y., Ponomarenko, A., Logvinov, A., and Krylov, V. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45: 61–68, 2014.

Malkov, Y. A. and Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

May, A. and Ozerov, I. On computing nearest neighbors with applications to decoding of binary linear codes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 203–228. Springer, 2015.

Meiser, S. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

Motwani, R., Naor, A., and Panigrahy, R. Lower bounds on locality sensitive hashing. *SIAM Journal on Discrete Mathematics*, 21(4):930, 2007.

Navarro, G. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28–46, 2002.

O'Donnell, R., Wu, Y., and Zhou, Y. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory (TOCT)*, 6 (1):5, 2014.

Pennington, J., Socher, R., and Manning, C. D. GloVe: global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Sablayrolles, A., Douze, M., Schmid, C., and Jégou, H. Spreading vectors for similarity search. In *International Conference on Learning Representations*, 2018.

Shakhnarovich, G., Darrell, T., and Indyk, P. *Nearest-neighbor methods in learning and vision: theory and practice (neural information processing)*. The MIT press, 2006.

Valiant, G. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *Journal of the ACM (JACM)*, 62(2):13, 2015.

Wang, J., Wang, J., Zeng, G., Tu, Z., Gan, R., and Li, S. Scalable k-NN graph construction for visual descriptors. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1106–1113, 2012.

Watts, D. J. and Strogatz, S. H. Collective dynamics of small-worldnetworks. *Nature*, 393:440442, 1998.

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Philip, S. Y., et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.